-1-

SIMULATION PROGRAMMING USING SIMSCRIPT II

Philip J. Kiviat*

The RAND Corporation, Santa Monica, California

The simulation described in this paper has been designed to illustrate as much of SIMSCRIPT II as possible in a natural, problem-oriented setting. While the program uses all of the language's features that are important in simulation studies, it does not use every SIMSCRIPT feature. A complete description is contained in P. J. Kiviat and R. Villanueva, The SIMSCRIPT II Programming Language, R-460-PR, The RAND Corporation, September 1968.

Even though the features illustrated are not exhaustive, the example may still seem forced and artificial. This should not be surprising, as it is a rare program that requires the full facilities of any complex programming language. The example is an elaboration of the job shop model of Chapter 3 of the SIMSCRIPT II report.**

The plan of the paper is as follows: the first section describes a model of a system in general terms, presents some problems the model has been designed to study, and places the rest of the paper in perspective. The next section contains a listing of the complete simulation program followed by a set of typical data cards. The last section works through the program section by section and, occasionally, where it is warranted, statement by statement, explaining the syntax and semantics of the statements.

H. M. Markowitz, B. Hausner and H. W. Karr, SIMSCRIPT - A Simulation Programming Language, Prentice-Hall Inc., New York, 1963.

THE SYSTEM

The system under study is shown abstractly in Fig. 1. It is a shop containing N production centers, each containing M_i identical machines, and finished goods inventory storage area. The shop produces P standard products for local sale and distribution, and variations of standard products for local and export distributors. Each product ordered travels through the shop, undergoing processing at production centers according to standard routings, production times and expediting procedures.
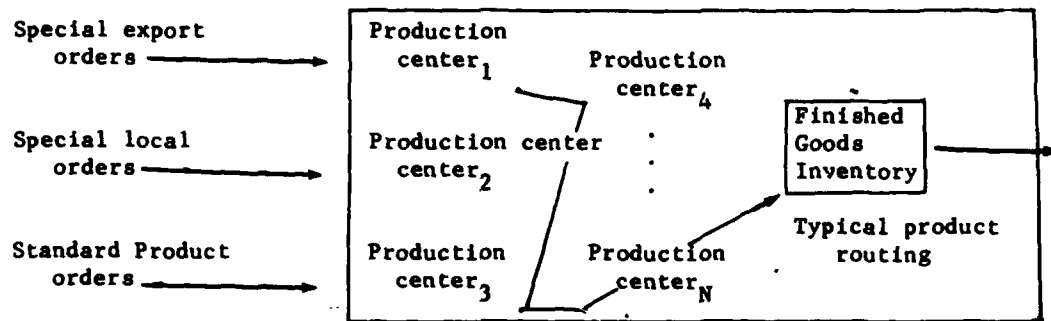

Fig. 1 -- System Under Investigation

Each production center has an in-process inventory area where products in process are stored if they cannot be worked on when they arrive. To minimize the value of in-process inventory, the production rules of'the shop remove partially completed products from production center queues according to a "high value first" rule.

Table 1 shows the entity-attribute-set model of the shop and its product line. These descriptors explain the shop's static structure. Permanent entities are used for production centers and product descriptions. Temporary entities are used for jobs and for job processing specifications.

The shop operates roughly as follows. When orders for standard products come into the shop, a standard production sequence is copied from an order book onto a job's production routing tag. The job is sent to the first production center, where it is worked on if a

machine is free. If no machines are available, the job is put in a work-in-process queue until a machine becomes available. When a job finishes processing, its routing tag is examined, and the job is either sent on to another processing center or to the finished goods inventory.

The shop's dynamic structure is captured in two events -- SALE and END.OF.PROCESS. Two other events, WEEKLY.REPORT and END.OF. SIMULATION, print periodic system performance data and stop the simulation. SALE is set up so it can be triggered both internally and externally. When triggered internally, SALE represents a local sale of a standard product. When triggered externally, SALE represents either a local or export sale of a special order. Two external event data tapes are provided to supply special order information. In SALE, jobs are assigned to machines and the system state changes to reflect such assignments. When a job is assigned to a machine, an END.OF.PROCESS event is scheduled to terminate the processing, make the machine available for another job, and pass the job on for additional processing or for shipment.

The simulation model is designed to determine the number of machines needed at each production center to provide "adequate" customer service. To study the effects of varying the number of machines in each center, a TALLY statement looks at the length of time jobs spend in the shop, and an ACCUMULATE statement looks at the waiting lines that build up at the production centers. Some number of machines will be chosen that balances the cost of degraded customer service with the costs of additional machines.

The program listing that follows has been written and annotated to make it as readable as possible. Statements that are not clear from the program itself are discussed in the section following the listing.

Table 1

ENTITIES, ATTRIBUTES AND SETS OF THE SHOP MODEL

Entity	Set or Attribute	Comment
PRODUCTION.CENTER The shop has N production centers.	NUMBER.IDLE	The number of idle machines in a production center
	QUEUE	Each production center owns a collection of in-process products.
PRODUCT The shop produces P different products.	SALES.FREQUENCY	Characterizes the frequency with which orders for a standard product arrive at the shop.
	NAME	Identifies a product
	STRUCTURE	Each product has a list of standard operations that have to be performed to produce it.
JOB Each order for a product is called a job.	VALUE	The dollar value of a job
	DUE.DATE	The time a job is promised to a customer
	ARRIVAL.TIME	The time a job is ordered
	EXPEDITE.FACTOR	The degree to which a job's processing can be speeded up at a production center.
	ROUTING	A list of production centers through which a job has to be processed
	FINISHED.GOODS.INVENTORY	Jobs can be placed in finished goods inventory awaiting shipment
THE SYSTEM The shop	FINISHED.GOODS.INVENTORY	Jobs are placed in finished goods inventory if finished before their due date.
OPERATION A task performed by a production center in producing a product.	MACHINE.DESTINED	The production center at which a job must be processed.
	CODE	A number representing a particular processing task.
	PROCESS.TIME	The time it takes to perform a processing task
	STRUCTURE	The standard parts list on which different operations appear
	ROUTING	The processing operations required for a particular job

```
**          SAMPLE SIMSCRIPT II SIMULATICN PROGRAM
**                   A JOB SHCP SIMULATION


PREAMBLE
NORMALLY MODE IS INTEGER AND CIMENSION IS 0

PERMANENT ENTITIES.....
        EVERY PRODUCT HAS A SALES.FRECUENCY ANC A NAME ANC OWNS A STRUCTURE
                CEFINE SALES.FREQUENCY AS A REAL RANDOM LINEAR VARIABLE
                DEFINE NAME AS AN ALPHA VARIABLE
        EVERY PRODUCT,PRODUCT HAS A PRODUCT.SALES(*/2)
        EVERY PRODUCTICN.CENTER HAS A (MAX.IN.CUEUE(1/2),MAX.QUEUE(2/2)) IN ARRAY 1,
            A (WNUM(1/2), MNUM(2/2)) IN ARRAY 2, A WSUM, A MSUM, A NUMBER.IDLE
            AND OWNS A QUEUE
                DEFINE NUMBER.IDLE AS A VARIABLE MONITORED ON THE LEFT


TEMPORARY ENTITIES.....
        EVERY JOB HAS A VALUE IN WORD 2, A CUE.DATE, AN ARRIVAL.TIME,
            AN EXPEDITE.FACTOR FUNCTICN, MAY BELONG TO A QUEUE, OWNS A ROUTING
            AND MAY BELONG TC THE WAITING.SET
                DEFINE EXPEDITE.FACTCR AS A REAL FUNCTION
                DEFINE VALUE, DUE.DATE AND ARRIVAL.TIME AS REAL VARIABLES
                DEFINE ROUTING AS A FIFO SET WITHOUT P AND N ATTRIBUTES
                DEFINE QUEUE AS A SET RANKED BY HIGH VALUE
        EVERY OPERATICN HAS A ( CODE(1/2) AND MACHINE.DESTINEC(2/2) ) IN WORD 1
            AND A PROCESS.TIME AND BELONGS TC A STRUCTURE AND A ROUTING
                DEFINE STRUCTURE AS A SET RANKED BY LCW CCDE WITHCUT M ATTRIBUTE
                        AND WITHOUT R ROUTINES
                DEFINE PROCESS.TIME AS A REAL VARIABLE


EVENT NOTICES INCLUCE WEEKLY.REPCRT
        EVERY SALE HAS A PRODUCT.TYPE, A PRICE ANC A PRIORITY
                DEFINE PRICE AS A REAL VARIABLE
        EVERY ENC.CF.PROCESS HAS AN ITEM ANC A PRCDUCER


BREAK SALE TIES BY HIGH PRICE THEN BY LOW PRICRITY
EXTERNAL EVENTS ARE ENC.OF.SIMULATION AND SALE
EXTERNAL EVENT UNITS ARE LOCAL.SALES AND EXPORT.SALES
PRICRITY ORDER IS END.CF.PROCESS, SALE, WEEKLY.REPORT AND END.OF.SIMULATION

BEFORE FILING AND REMOVING FRCM QUEUE CALL QUEUE.CHECK
BEFORE DESTROYING JOB, CALL STAY.TIME


                DEFINE STAY AS A REAL DUMMY VARIABLE
TALLY AVG.STAY AS THE WEEKLY MEAN, VAR.STAY AS THE WEEKLY VARIANCE, SUM.STAY AS
                THE WEEKLY SUM, SUM.SCUARES.STAY AS THE WEEKLY SUM.CF.SCUARES, AND
                NUM.STAY AS THE WEEKLY NUMBER OF STAY
ACCUMULATE WSUM AS THE WEEKLY SUM, WNUM AS THE WEEKLY NUMBER, AVG.QUEUE AS THE
                WEEKLY MEAN, MAX.QUEUE AS THE WEEKLY MAXIMUM AND FREC(0 TO 25 BY 1)
                AS THE WEEKLY HISTOGRAM OF N.QUEUE
ACCUMULATE MSUM AS THE MONTHLY SUM, MNUM AS THE MCNTHLY NUMBER, AVG.IN.QUEUE AS
                THE MONTHLY MEAN, MAX.IN.QUEUE AS THE MONTHLY MAXIMUM OF N.QUEUE


THE SYSTEM OWNS A FINISHED.GOODS.INVENTORY
DEFINE FINISHED.GOODS.INVENTORY AS A SET RANKED BY DUE.DATE
DEFINE LOCAL TC MEAN DEFINE I,J,K,L,M AND N AS SAVED INTEGER VARIABLES
DEFINE WEEK TO MEAN *HOURS.V*7 HOURS
DEFINE PRICRITY.FREQUENCY AS A 2-DIMENSICNAL ARRAY
```

```
DEFINE TITLE AS A TEXT VARIABLE
DEFINE WEEK.COUNTER AND TAPEJFLAG AS INTEGER VARIABLES
DEFINE AVERAGE AS A REAL FUNCTICN WITH 1 ARGUMENT
END



MAIN
'INITIALIZE'  PERFORM INITIALIZATION
LET BETWEEN.V='TRACE'    START SIMULATION


'' PERFORM NEXT EXPERIMENT ''
FOR EACH JOB IN FINISHED.GOODS.INVENTORY, DO
     REMOVE THE JOB FROM FINISHED.GOODS.INVENTORY
     DESTROY THE JOB
LOOP

FOR EACH PRODUCTION.CENTER,
FOR EACH JOB IN QUEUE, DO
    FOR EACH OPERATION IN ROUTING, DO
         REMOVE THE OPERATION FROM ROUTING
         DESTROY THE OPERATION
    LOOP
    REMOVE THE JOB FROM QUEUE
    DESTROY THE JOB
LOOP

FOR EACH PRODUCT, DO
    FOR EACH OPERATION IN STRUCTURE, DO
         REMOVE THE OPERATION FROM THE STRUCTURE
         DESTROY THE OPERATION
    LOOP
ALSO FOR EACH RANDOM.E IN SALES.FREQUENCY, DO
         REMOVE THE RANDOM.E FROM SALES.FREQUENCY
         DESTROY THE RANDOM.E
LOOP

RELEASE NAME, F.STRUCTURE,L.STRUCTURE, N.STRUCTURE, F.SALES.FREQUENCY,
    PRODUCT.SALES, NUMBER.IDLE, F.QUEUE, L.QUEUE, N.QUEUE, MAX.IN.QUEUE,
    MAX.QUEUE, WSUM, MSUM, WNUM, MNUM
RELEASE PRIORITY.FREQUENCY
ERASE TITLE

RESET TOTALS OF STAY
FOR EACH PRODUCTION.CENTER, RESET TOTALS OF N.QUEUE

LET WEEK.COUNTER=0
WRITE AS "@@@@",/
LET TAPEJFLAG=0

'' REUSE EXTERNAL EVENTS IN NEXT EXPERIMENT
REWIND LOCAL.SALES AND EXPORT.SALES

GO INITIALIZE
STOP    END
```

```
ROUTINE FOR INITIALIZATION
LOCAL
DEFINE PF TO MEAN PRIORITY.FREQUENCY
DEFINE SF TO MEAN SALES.FREQUENCY
DEFINE CHECK AS AN ALPHA VARIABLE
LET EOF.V=1
INPUT TITLE
IF EOF.V=2, PRINT 1 LINE AS FOLLOWS
            ENC OF CATA HIT
        STOP
ELSE

REAC N.PRODUCTION.CENTER
    CREATE EVERY PRODUCTION.CENTER
    FOR EACH PRODUCTION.CENTER, REAC NUMBER.ICLE

REAC N.PRODUCT
    CREATE EVERY PRODUCT
    RESERVE PRIORITY.FREQUENCY(*,*) AS N.PRODUCT BY *
    FOR EACH PRODUCT, DO
            READ NAME
            READ SALES.FREQUENCY
            RESERVE PRIORITY.FREQUENCY(PRODUCT,*) AS PRODUCT
            FOR I=1 TO PRODUCT, READ PRIORITY.FREQUENCY(PRODUCT,I)
            UNTIL MODE IS ALPHA, DO THIS.....
                    CREATE AN OPERATION
                    FILE THE OPERATION IN STRUCTURE
                    READ CODE, MACHINE.DESTINED AND PROCES .TIME
            LOOP
            SKIP 1 FIELD
            CAUSE A SALE IN SF HOURS
            LET PRODUCT.TYPE=PRODUCT
            LET PRICE=PRODUCT*RANDOM.F(1)
            LET PRIORITY=PF(PRODUCT, TRUNC.F(PRICE)+1)
    LOOP

REAC LOCAL.SALES, EXPORT.SALES AND SAVE.TAPE

REAC MONTH, DAY AND YEAR    CALL ORIGIN.R(MONTH,DAY AND YEAR)

REAC CHECK   IF CHECK EQUALS "OK", CALL REPORT   RETURN
                OTHERWISE   PRINT 1 LINE AS FOLLOWS
    EITHER TOO MUCH DATA OR DATA HAS BEEN READ INCORRECTLY
STOP   END



EVENT SALE(PRODUCT,PRICE,PRIORITY) SAVING THE EVENT NOTICE
DEFINE SF TO MEAN SALES.FREQUENCY
LOCAL
IF SALE IS EXTERNAL, READ PRODUCT, PRICE AND PRIORITY AS B 30,I 5, D(10,3), I 5
REGARDLESS   ADD 1 TO PRODUCT.SALES(PRODUCT, TRUNC.F(PRICE)+1))
CREATE A JOB
    LET VALUE=PRICE
    LET DUE.DATE=TIME.V + PRICE + PRIORITY
    LET ARRIVAL.TIME=TIME.V
IF SALE IS INTERNAL,
    FOR EACH PIECE OF STRUCTURE, FILE PIECE IN ROUTING   GO TO JOB
** PROCESS SPECIAL ORDERS
```

```
OTHERWISE UNTIL MODE IS ALPHA, DO THE FOLLOWING...
                READ N
                FOR EACH  PIECE IN STRUCTURE WITH  CODE(PIECE) = N,
                FIND THE FIRST CASE, IF NONE GO TO LOOP
                FILE PIECE IN ROUTING
   'LOOP'   LOOP
'JOB'   NOW ATTEND.TO.JOB
IF SALE IS EXTERNAL, DESTROY THE SALE   RETURN
OTHERWISE......
   SCHEDULE THE SALE(PRODUCT, PRODUCT*RANDOM.F(1), PRIORITY.FREQUENCY(PRODUCT,
           TRUNC.F(PRICE)+1) IN SF HOURS
RETURN   END



ROUTINE TO ATTEND.TO.JOB
LET PRODUCTION.CENTER=MACHINE.DESTINED(F.ROUTING(JOB))
IF NUMBER.IDLE IS POSITIVE,
    SUBTRACT 1 FROM NUMBER.IDLE    PERFORM ALLOCATION
    RETURN
OTHERWISE   FILE JOB IN QUEUE    RETURN
END



ROUTINE FOR ALLOCATION
REMOVE THE FIRST OPERATION FROM THIS ROUTING
SCHEDULE AN END.OF.PROCESS GIVEN JOB AND PRODUCTION.CENTER IN
    PROCESS.TIME*EXPEDITE.FACTOR HOURS
RETURN   END



ROUTINE EXPEDITE.FACTOR
IF TIME.V IS GREATER THAN DUE.DATE RETURN WITH 0.5   ELSE
RETURN WITH MIN.F((DUE.DATE-TIME.V)/PROCESS.TIME,1)
END



UPON END.OF.PROCESS GIVEN JOB AND PRODUCTION.CENTER
IF ROUTING IS EMPTY, IF DUE.DATE <= TIME.V,
                        DESTROY THIS JOB   GO TO PC
                    ELSE FILE THIS JOB IN FINISHED.GOODS.INVENTORY   GO TO PC
OTHERWISE  CALL ATTEND.TO.JOB
'PC'
IF QUEUE IS EMPTY,
    ADD 1 TO NUMBER.IDLE    RETURN
ELSE   REMOVE THE FIRST JOB FROM QUEUE
PERFORM ALLOCATION   RETURN
END



EVENT FOR WEEKLY.REPORT SAVING THE EVENT NOTICE
RESCHEDULE THIS WEEKLY.REPORT IN 1 WEEK
ADD 1 TO WEEK.COUNTER
NOW REPORT
RESET WEEKLY TOTALS OF STAY
```

```
FOR EACH PRODUCTION.CENTER, RESET WEEKLY TOTALS OF N.CUEUE
IF MOD.F(WEEK.COUNTER,4)=C, FOR EACH PRODUCTICN.CENTER, RESET MONTHLY TOTALS
     CF NJQUEUE    ELSE
RETURN    END



EVENT FOR END.OF.SIMULATION
FOR I=1 TC EVENTS.V, FCR EACH NCTICE IN EV.S(I), DO
     REMOVE THE NOTICE FROM EV.S(I)
     DESTROY THE NOTICE
LCOP
NOW REPORT
LIST PRODUCT.SALES
RETURN    END



ROUTINE FOR QUEUE.CHECK GIVEN ENTITY AND I
LCCAL
IF 1 LE I LE N.PRODUCTION.CENTER, RETURN
OTHERWISE... PRINT 1 LINE WITH I AS FOLLOWS
     STOPPEO TRYING TO REFERENCE CUEUE(    *)
TRACE   STOP   END



ROUTINE FOR STAY.TIME GIVEN JOB
LET STAY.TIME.V - ARRIVAL.TIME(JCB)
RETURN    END



RCUTINE TC TRACE
LCCAL
IF FINISHED.GCCDS.INVENTORY IS EMPTY, GO AROUND
ELSE  FOR EACH JOB IN FINISHED.GOODS.INVENTORY UNTIL DUE.DATE > TIME.V, DO
         REMOVE THE JCB FROM FINISHED.GCODS.INVENTORY  DESTROY THE JOB
      LCOP
'AROUNC'
GC TC END.CF.PROCESS, SALE, WEEKLY.REPORT AND END.OF.SIMULATION PER EVENT.V
'END.CF.PROCESS'  WRITE ITEM, PRODUCER, TIME.V AS "ITEM", I 5, "STOPPED",
                     "PROCESSING CN MACHINE", I 5, " AT TIME=",D(10.3), /
             RETURN
'SALE'  WRITE TIME.V, PRODUCT.TYPE, PRICE AND PRIORITY AS "SALE OF TYPE", I 3, "
            " PRODUCT AT TIME=", D(10,3), " FOR $", D(6 2), " PRIORITY=", I 3,
             /    RETURN
'WEEKLY.REPORT'
'END.CF.SIMULATION'
RETURN
END



LEFT ROUTINE NUMBER.IDLE(M)
CEFINE J AS A SAVED, 2-CIMENSICNAL ARRAY
CEFINE K AS A SAVEC, 1-CIMENSICNAL ARRAY
ENTER WITH N
IF TAPE.FLAG=0,  LET TAPE.FLAG=1
```

```
                    RELEASE J AND K
                    RESERVE K(*) AS N.PRODUCTION.CENTER
                    RESERVE J(*,*) AS 100 BY N.PRODUCTION.CENTER
REGARDLESS  ADD 1 TO K(M)
LET J(K(M),M)=N
IF K(M)=100, WRITE M AS I 3 USING SAVE.TAPE
            FOR I=1 TO 100, WRITE J(I,M) AS I 3 USING SAVE.TAPE
            WRITE AS / USING SAVE.TAPE
REGARDLESS  MOVE FROM N
RETURN    END



ROUTINE TO REPORT
LOCAL
IF TIME.V=0,
    START NEW PAGE  ''AND''  OUTPUT TITLE
    SKIP 2 OUTPUT LINES
    PRINT 3 LINES AS FOLLOWS
                    P R O D U C T   D A T A
        NAME         SALES FREQUENCY      PRODUCTION SEQUENCE
                     PROB.    VALUE    CODE CNTR   TIME
    FOR EACH PRODUCT, DO
      LET I=F.SALES.FREQUENCY
      LET J=F.STRUCTURE
      PRINT 1 LINE WITH NAME, PROB.A(I), RVALUE.A(I), CODE(J),
                    MACHINE.DESTINED(J) AND PROCESS.TIME(J) THUS
        ****        *.**       *.**      **    **    *.**
      IF I¬=0, LET I=S.SALES.FREQUENCY(I)  ELSE
      IF J¬=0, LET J=S.STRUCTURE(J)  ELSE
      IF I=0 AND J=0, GO TO 'LOOP'  ELSE
      IF I¬=0 AND J=0, PRINT 1 LINE WITH PROB.A(I) AND RVALUE.A(I) THUS
            *.**       *.**
      ELSE IF I=0 AND J¬=0, PRINT 1 LINE WITH CODE(J), MACHINE.DESTINED(J),
                                    AND PROCESS.TIME(J) THUS
                        **    **    *.**
      ELSE IF I¬=0¬=J, PRINT 1 LINE WITH PROB.A(I), RVALUE.A(I), CODE(J),
                            MACHINE.DESTINED(J) AND PROCESS.TIME(J) THUS
            *.**       *.**      **    **    *.**
    'LOOP' LOOP
    SKIP 2 OUTPUT LINES
    PRINT 2 LINES AS FOLLOWS
            P R O D U C T I O N   C E N T E R   D A T A
                CENTER        NUMBER OF MACHINES
    FOR EACH PRODUCTION.CENTER, PRINT 1 LINE WITH PRODUCTION.CENTER AND
                            NUMBER.IDLE THUS
                **                  **
    SKIP 2 OUTPUT LINES
    PRINT 2 LINES AS FOLLOWS
                    I N I T I A L   E V E N T S
                    EVENT TYPE      TIME
    FOR I=1 TO EVENTS.V, FOR EACH J IN EV.S(I),
    PRINT 1 LINE WITH I AND TIME.A(I) THUS
                    *           **.**
REGARDLESS.....
START NEW PAGE
PRINT 1 LINE AS FOLLOWS
PRINT 3 LINES LIKE THIS.....
                W E E K L Y   R E P O R T
```

```
PRINT 2 LINES WITH AVG.STAY AND VAR.STAY AS FOLLOWS
   JOB STAY STATISTICS ARE:  AVERAGE STAY= *.**
                             VARIANCE    = *.**
SKIP 3 OUTPUT LINES
BEGIN REPORT
BEGIN HEADING
PRINT 2 LINES AS FOLLOWS
     PRODUCTION CENTER QUEUEING REPORT
         CNTR      AVG. QUEUE     MAX. QUEUE
END  **HEADING
FOR EACH PRODUCTION.CENTER, PRINT 1 LINE WITH PRODUCTION.CENTER,
                             AVG.QUEUE AND MAX.QUEUE THUS
          **           *.**              *
END  ** REPORT
PRINT 1 LINE WITH AVERAGE(AVG.QUEUE(*)) LIKE THIS
     OVERALL AVERAGE QUEUE LENGTH OF ALL QUEUES IS  *.**
SKIP 3 OUTPUT LINES
FOR EACH PRODUCTION.CENTER, DO
     BEGIN REPORT PRINTING FOR I=1 TO 25 IN GROUPS OF 5
     BEGIN HEADING
       PRINT 1 LINE WITH PRODUCTION.CENTER LIKE THIS
     HISTOGRAM OF QUEUE LENGTH FOR PRODUCTION CENTER  **
     END ** HEADING
     PRINT 1 LINE WITH A GROUP OF FREQ(PRODUCTION.CENTER,2) FIELDS THUS
          *    *    *    *    *
     END ** REPORT
LOOP
IF MOD.F(WEEK.COUNTER,4)-=0, RETURN
OTHERWISE...  START NEW PAGE
PRINT 1 LINE AS FOLLOWS
          M O N T H L Y   R E P O R T
SKIP 2 OUTPUT LINES
SKIP 3 OUTPUT LINES
BEGIN REPORT
BEGIN HEADING
PRINT 2 LINES AS FOLLOWS
     PRODUCTION CENTER QUEUEING REPORT
         CNTR      AVG. QUEUE     MAX. QUEUE
END  **HEADING
FOR EACH PRODUCTION.CENTER, PRINT 1 LINE WITH PRODUCTION.CENTER,
                             AVG.IN.QUEUE AND MAX.IN.QUEUE THUS
          **           *.**              *
END  ** REPORT
PRINT 1 LINE WITH AVERAGE(AVG.IN.QUEUE(*)) LIKE THIS
     OVERALL AVERAGE QUEUE LENGTH OF ALL QUEUES IS  *.**
RETURN
END


ROUTINE FOR AVERAGE GIVEN ARRAY
LOCAL
DEFINE ARRAY AS A 1-DIMENSIONAL ARRAY
FOR J=1 TO DIM.F(ARRAY(*)), COMPUTE M AS THE MEAN OF ARRAY(J)
RETURN WITH M
END
```

```
SAMPLE DATA FOR SEVERAL JOB SHOP EXPERIMENTS USING THE LEVEL K, SECTION 5.05
JOB SHOP SIMULATION PROGRAM.J...TITLES, AS SHOWN, CAN EXTEND OVER SEVERAL CARDS
AND ARE ENDED BY A MARK.V CHARACTER      *
5  1C 10 5 5 3
3
TOP   C.25 10.0    0.50 15.0    0.75 20.0    1.00 25.0
1      1 1 C.2    2 2 0.5    3 4 0.3*
YCYC      C.10 3.7    0.28 5.6    0.39 7.2    0.60 9.2    0.81 10.6    0.95 15.2
          1.00 20.0*
1 2      15 1 1.2    16 3 0.8    17 5 0.2    18 2 1.5*
CCLL      0.1 1.0    0.2 2.0    0.2 3.0    0.4 4.C    0.1 5.0
1 2 3      20 5 4.2    21 4 5.6    22 1 3.2    23 5 2.00*

1  2  3
7 1 1968
CK
THIS IS A TITLE CARD FOR THE SECOND SIMULATION EXPERIMENT OF THE SERIES
DATA CARDS FOR THIS EXPERIMENT WILL HAVE THE SAME FORMAT AS THOSE OF THE
PREVIOUS EXPERIMENT AND WILL END WITH AN "OK" CARD    *


THE FOLLOWING CARDS ARE SAMPLES OF THE DATA CARDS PUNCHED FOR ONE OF THE TWO
EXTERNAL EVENTS TAPES     ..JJ........... THIS IS JUST A SAMPLE FROM THE TAPE

SALE 7/2/68   12 CC    2 1.98 1    15 17 18 *
SALE 7/2/68   13 25    1 0.97 1    1  2  3 *
SALE 7/3/68   01 30    2 1.50 2    16 17 18 *
SALE 7/3/68   C7 00    3 2.50 1    20 22 23 *
END.CF.SIMULATION 9/1/69   12 00 *
```

COMMENTS ON THE SIMULATION PROGRAM

The program is arranged functionally and is discussed as it appears. The order of the program is preamble, main routine, initialization, events and routines of the simulation model, routines for monitoring, debugging and analysis.

Preamble

The preamble is divided into seven sections: permanent entities, temporary entities, event notices, event control, debugging, analysis and miscellaneous declarations. Most simulation programs can be organized this way.

One compound and two simple permanent entities are declared. The special features of each are these:

PRODUCT has a RANDOM attribute and an ALPHA attribute, each requiring definition in a DEFINE statement.

PRODUCT.SALES, the single attribute of the compound entity PRODUCT,PRODUCT, is intrapacked to conserve storage space.

PRODUCTION.CENTER has two pairs of attributes that are packed in the same array, and one attribute that is monitored. The packed attributes use field-packing, equivalence and array specification. The monitored attribute requires an additional DEFINE statement.

PRODUCT.SALES could just as easily have been defined as a global array or as a two-dimensional system attribute. As a global array though, it could not have been packed; as a system attribute it would not be eligible for implied subscripting.

Two temporary entities are declared. The special features of each follows:

JOB has one attribute placed in a specific word in its entity record, and has a function attribute. The function attribute requires a DEFINE statement to declare its mode.

Two sets in which a JOB participates have their implied properties modified by DEFINE statements.

Two attributes of OPERATION are packed in the first word of each entity record.

A set to which an OPERATION belongs has its removal routines
deleted by a DEFINE statement.

Three event notices are declared. The special features of each
are these:

WEEKLY.REPORT has no attributes, and neither owns nor belongs
to sets other than the standard one defined for all event
notices.

One event, SALE, breaks ties among competing event notices
through a BREAK TIES declaration. The other internal
events break ties, if they occur, on a first-come, first-
served basis.

Two external event classes, END.OF.SIMULATION and SALE, are declared.
Two input devices are declared as suppliers of external event triggers.
The priority order of the four event classes is declared in a PRIORITY
statement.

Two BEFORE statements are used. Each states that a certain
routine is to be called before a specified action takes place. The
arguments of these routines are not stated, but implied.

One TALLY and two ACCUMULATE statements are used. The special
features of each follow:

The TALLY statement compiles statistics for a DUMMY variable,
which is declared in a separate DEFINE statement.

All of the statistical counters used in the TALLY and
ACCUMULATE statements are defined so they can be released.
If they were not named, they would be given local names
such as A.1 and A.2 by the SIMSCRIPT II compiler.

FREQ is a two-dimensional array. The first dimension is
an entity index. The variable for which it accumulates
a histogram is an attribute of PRODUCTION.CENTER. The
second dimension is the histogram index and is an
integer between 1 and 26.

The remaining statements are:

Declare a system owned set.

Use DEFINE TO MEAN statements to create shorthand notation.

Declare four global variables: a two-dimensional array,
two INTEGER variables, and a TEXT variable.

Declare a function and specify the number of arguments it must always have.

Main Program

The main routine has three functions: it initializes the model so simulation can start, it transfers control to the timing routine when initialization is complete, and it resets the entire system to an "empty" condition at the end of a simulation run so another experiment can begin.

Initialization takes place in the routine INITIALIZATION. After initialization, the SUBPROGRAM system variable BETWEEN.V is set to the routine name 'TRACE', indicating that this routine is to be called before each event is executed. Simulation begins at the START SIMULATION statement that removes the first event from the events list and transfers control to it. Simulation proceeds until an END.OF. SIMULATION event occurs. This event, aside from its obvious task of reporting the results of the simulation experiment, empties the events list sets. When END.OF.SIMULATION returns control to the timing routine, the lack of scheduled events causes control to pass to the statement after START.SIMULATION. In many simulations this will be STOP. In this example, it is the first of many statements that release and destroy all permanent and temporary entities. After these statements have been executed, all the memory structures set up by the previous experiment have been erased.

To perform this erasure, the system set FINISHED.GOODS.INVENTORY and the sets owned by the permanent and temporary entities are emptied and their members destroyed. Finally, all attributes of permanent entities are released. Special features to notice are these:

The PRODUCTION.CENTER loop in which operations owned by jobs owned by production centers are successively removed and destroyed.

The RANDOM variable SALES.FREQUENCY is treated as a set when it is emptied.

All permanent entity attributes, including set pointers and statistical accumulators, are released.

In many programs, so extensive a reinitialization process is not necessary. For example, it is usually sufficient to zero out all attribute values and empty all sets. This example has been written to illustrate what seems to be the worst case. When single simulation runs are made and no reinitialization is necessary, the initialization routine can be released and its space regained for array and entity storage. The following routine shows how this is done.

Add to the preamble:

DEFINE INITIALIZATION AS A RELEASABLE ROUTINE

Use this routine:

```
MAIN
PERFORM INITIALIZATION
RELEASE INITIALIZATION
START SIMULATION
STOP     END
```

Program Initialization

INITIALIZATION starts with some declarations. The first takes advantage of the DEFINE TO MEAN statement of the preamble to define some local INTEGER variables I, J, K, L, M and N. The next two statements are local DEFINE TO MEAN declarations that create short-hand notations for two lengthy variable names. The last declaration describes a local ALPHA variable that verifies whether or not all input data have been read.

Since a mistake may have been made in setting up a simulation run, EOF.V is set to 1 to give the program control over the actions taken when the end of the input data file is reached. If an end-of-file is encountered when reading TITLE, EOF.V is set to 2 and this fact is picked up in the following IF statement. A sequence of simulation experiments can also be stopped this way. When all the data for a sequence of runs are exhausted, these statements will stop the program.

The INPUT statement reads characters from the current input unit READ.V until an asterisk, the MARK.V default symbol, is reached. A typical simulation TITLE card might be:

SIMULATION RUN NO. 1 JOB SHOP WITH 10 CENTERS   *

If some symbol other than * is to be used as a TEXT terminator, a statement such as LET MARK.V="?" is put at the head of INITIALIZATION.

A value that is the number of production centers is then read and used to reserve the arrays that hold the attributes of PRODUCTION. CENTER. This value is also used to read in the number of machines in each production center (which are all idle when simulation begins.)

A similar process then takes place for PRODUCT. It is more complex in that a richer variety of data structures are associated with PRODUCT than with PRODUCTION.CENTER. The data structures are the following:

> PRIORITY.FREQUENCY--a "ragged table" whose rows are "sized" dynamically.

> SALES.FREQUENCY--a RANDOM variable whose sampling data is read in a standard format.

> STRUCTURE--a set with OPERATIONS as members.

Also, an initial local SALE for a standard product must be scheduled for each product type. In scheduling these sales, the PRICE of each SALE is a random variable between 0 and the product type, e.g., a type 3 product can be sold for between $0 and $3, and the PRIORITY assigned to a sale can be determined by a random draw from the PRIORITY.FREQUENCY table.

At the end of initialization the numbers of the input devices for the LOCAL.SALES and EXPORT.SALES external event units are read. This allows devices to be changed between simulation runs. Finally, the ORIGIN.R routine is invoked to set the simulation calendar so that calendar dates can be used on the external event tapes.

If the last data field read is not the character string OK, the run terminates with an error message.

Events and Routines of the Simulation Model

The event SALE is written to react properly to both internal
and external event triggers. The event creates a job, determines
its routing through the shop, and starts it in its processing sequence.
If the sale is internal, a new order is scheduled for the same product
some time in the future.

The EVENT statement defines SALE as an event routine with three
input arguments. It also declares that when a SALE event notice is
selected as the next event, the first three programmer defined
attributes of SALE are to be assigned to the local variables,
PRODUCT, PRICE and PRIORITY, and that the event notice is not to be
destroyed. An important point to note here is that PRODUCT, PRICE
and PRIORITY are local variables; they are different from the variables
defined as attributes of the event notice in the preamble.

The first two statements are local declarations that we have
seen before. The third statement is the one that allows SALE to
be used with both internal and external event triggers. It says:
if SALE has occurred externally, read three data items, otherwise
do not read any data. Regardless of how the event was triggered,
values get assigned to PRODUCT, PRICE and PRIORITY.

The next statement adds 1 to an element of PRODUCT.SALES, counting
the number of times particular products are sold at different prices.

The next section of code creates a JOB entity and assigns values
to its attributes. The JOB is the entity that will flow through the
shop, and will represent the sale from now on. If SALE is triggered
internally, it is a sale for a standard product, and the standard
sequence of operations for producing that product is transferred from
STRUCTURE(PRODUCT) to ROUTING(JOB). Notice that implied subscripts
are used in the program for both STRUCTURE and ROUTING. If the SALE
is triggered externally, it represents a possible special order. As
special order operations are subsets of operations that produce
standard products, data are read that select a subset of operations
for producing the order and store it in the JOB routine set.

With this, the task of SALE is almost completed. The now completely
specified JOB is started through processing by the routine ATTEND.TO.JOB.
After this routine deals with the job, it returns to SALE where
arrangements are made for the next SALE. If the SALE just processed
was triggered externally, the event notice is destroyed and control
returned to the timing mechanism. The next external event will be
selected automatically by the timing mechanism from the external
events tapes. If the SALE was triggered internally, the event notice
is reused to schedule another sale for this same product, with a
different price and priority. The time at which this sale is to
occur is determined by a random sample from the RANDOM variable
SALES.FREQUENCY(PRODUCT); again implied subscripting is used.

ATTEND.TO.JOB uses the routing of the current JOB to select
the production center in which the first operation is to be performed.
The first statement in ATTEND.TO.JOB is important as it illustrates
several basic operations in SIMSCRIPT II programming. First, it
illustrates the use of a set pointer to select a set member; F.ROUTING
(JOB) picks out an entity identification number. This identification
number is then used with an attribute of the entity type it represents
(OPERATION) to determine a value; MACHINE.DESTINED(R.ROUTING(JOB)) is
the production center number the first operation requires. This
number is assigned to PRODUCTION.CENTER so that implied subscripting
can be used later on in the program.

After the production center is determined, NUMBER.IDLE(PRODUCTION.
CENTER) is examined to determine whether a machine is available in this
production center to process the job. If a machine is available, it
is allocated to the job by first subtracting 1 from the idle machine
counter and then calling the routine ALLOCATION. If a machine is
not available, the job is filed in a QUEUE owned by the
production center. When a machine becomes free at some later date,
the QUEUE will be examined and a job removed for processing. All
the time a job spends in the shop in excess of its actual processing
time is spent in queues belonging to different production centers
and in finished goods inventory.

ALLOCATION knows that it is given a certain job because the
that the identification number of this job is in the global variable
JOB. It removes the first operation from the ROUTING of this job,
and schedules an END.OF.PROCESS event. The job identification
and the production center identification are assigned to the first
two attributes of this event notice. The event is scheduled for a
standard time, PROCESS.TIME(OPERATION), modified by a factor
that depends upon the current time and the time the job was
promised to the customer. Note the use of implied subscripts in
communicating between the routines ALLOCATION and EXPEDITE.FACTOR,
and within the routines themselves.

EXPEDITE.FACTOR looks at the DUE.DATE of the current job and
compares it with the current simulation time. If the job is late,
EXPEDITE.FACTOR returns a value of 0.5 to shorten the processing time
of the operation. If the job is not late, a value between 0 and 1 is computed,
which depends upon the time remaining before the job will be late
and the processing time of the current operation. Again
implied subscripts are employed.

The event END.OF.PROCESS does two things, it takes care of a
job that has just finished processing at a production center and it
takes care of the production center. First it deals with the job.
If the ROUTING of the job is empty, all its operations have been
completed and it can pass from the shop. If the job is finished on
time, or is late, its entity record is destroyed. If it is early,
it is filed in FINISHED.GOODS.INVENTORY where it stays until its
DUE.DATE. If the job is not completed, the routine ATTEND.TO.JOB
assigns it to its next operation. Note the use of the REMOVE state-
ment in ALLOCATION that makes this assignment automatic.

It is important at this point to understand the flow of control
between events and routines. The reader is advised to make up some
data, or use the data at the end of the program to trace several
jobs and their flow through the shop.

After disposing of the job, END.OF.PROCESS deals with the produc-
tion center. If no jobs are awaiting processing in the production

center QUEUE, the machine just released is returned to the idle state.
If jobs are waiting, one is selected according to the queue's priority
rule and processing is started on it.

This completes the routines and events that are directly involved
in the shop simulation. The remaining routines and events deal with
preparing reports, stopping the simulation, collecting data and
monitoring the model.

Events and Routines for Monitoring, Debugging and Analysis

The event WEEKLY.REPORT occurs periodically. It keeps track
of the number of simulated weeks that have gone by, resets counters
that are used for collecting periodic statistical information, calls
on a report routine and reschedules itself. The feature of interest
in this routine is the use of the word WEEK in the RESCHEDULE state-
ment. This word was defined to mean *HOURS.V*7 HOURS in the program
preamble; the RESCHEDULE statement is therefore compiled as RESCHEDULE
THIS WEEKLY.REPORT IN 1*HOURS.V*7 HOURS. When declarations such as
this are made, care must be taken that the defined word is not used
inadvertently in another context, e.g. there can be no variable,
routine, label or set named WEEK in this program.

The event END.OF.SIMULATION is triggered externally and has as
its main purpose the termination of a simulation run. It does this
by emptying the events sets so control will pass to the statement
after START SIMULATION when END.OF.SIMULATION returns control. In
addition to stopping the simulation, END.OF.SIMULATION calls REPORT
and lists an array.

The next two routines, QUEUE.CHECK and STAY.TIME, are associated
with the BEFORE statements of the preamble. QUEUE.CHECK is called
before filing or removing is done in any QUEUE; STAY.TIME is called
before any JOB is destroyed. The sole purpose of QUEUE.CHECK is
error checking; its code is straightforward. STAY.TIME has a
different purpose. It computes the time a job spends in the shop,
assigning this time to a variable that will have a TALLY operation.
Note that STAY is used to compute statistics through

its TALLY computations, but as it is not used anywhere else in the program, it is declared DUMMY and given no storage location.

TRACE is more complicated. Triggered by a call from BETWEEN.V before every event, it is used to trace events and to release jobs from finished goods inventory when the simulation clock reaches their due date. The first part of the routine deals with this task. The code makes use of the fact that jobs are ranked in FINISHED.GOODS. INVENTORY in order of their DUE.DATE.

The trace section of the program uses EVENT.V to branch to a different output statement according to the type of event that has been selected to occur next. These output statements print different items of information about each event type. The program could as easily be written to take actions on the event types, such as turning off the trace by setting BETWEEN.V=0 when TIME.V reaches a certain value or a special kind of event occurs.

Routine NUMBER.IDLE is a left-handed routine that implements the monitoring of the attribute NUMBER.IDLE. It has several unusual features. One reason for defining NUMBER.IDLE as a left-hand monitored variable is to save values of the number of machines idle over time for later processing, without putting the code to do this in the simulation routines. To remove this feature from the program at some later date, one need only remove the preamble card that states that NUMBER.IDLE is monitored and the routine NUMBER.IDLE, and recompile the program. No changes need be made to any other routines.

The program uses two SAVED local arrays to collect and write on tape successive values of NUMBER.IDLE for each production center. A global variable TAPE.FLAG is used to tell the routine when initialization of the SAVED local variables is required; TAPE FLAG is set to zero at the start of every simulation experiment. The routine demonstrates SAVED values, local arrays, a left-handed function, subscripted subscripts, and monitored variables.

The last routines deal with reports of system activity during a simulation experiment. They print out the parameters of the experiment and the measurements made during the experiment. They illustrate the report generating facilities of SIMSCRIPT II as well as the COMPUTE statement.